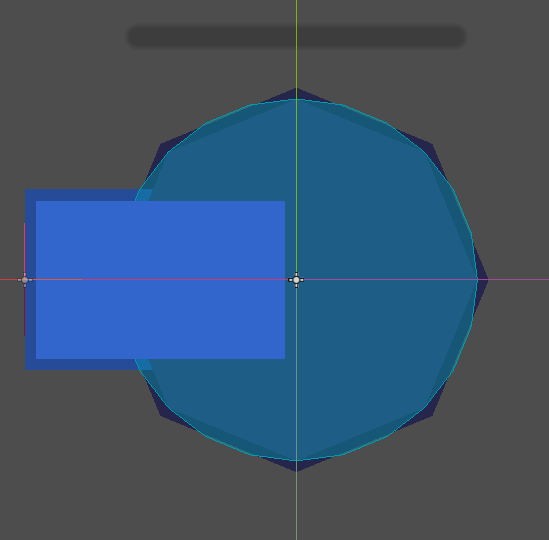
Учебная практика – 2 игра Top-Down shooter

Шаг 1: Создание узла AISpawner

Узел AISpawner, который отвечает за спавн AI-танков в игре. Добавил переменные: MaxAITanks = 5 — максимальное количество AI-танков, которые могут быть одновременно на поле. SpawnInterval = 5.0 — интервал спавна AI-танков в секундах. MinSpawnDistance = 800.0 — минимальное расстояние от игрока, на котором могут появляться AI-танки. MaxSpawnDistance = 1500.0 — максимальное расстояние от игрока для спавна AI-танков. Подключил скрипт AISpawner.cs, который управляет логикой спавна AI-танков.

Шаг 2: Создание AI-танка

Узел AITank, который представляет собой AI-танк. Подключил сцену Tank.tscn как базовый узел для AI-танка. В классе AITankController добавил переменные: DetectionRange = 1000.0 — радиус, в котором AI-танк может обнаружить игрока. PreferredCombatDistance = 500.0 — расстояние, на котором AI-танк предпочитает вести бой. WanderRadius = 400.0 — радиус блуждания AI-танка. AIUpdateInterval = 0.2 — интервал обновления логики AI.



Шаг 3: Настройка тела танка

Узел TankBody, который представляет собой визуальную часть танка. Установил цвет modulate = Color(1, 0.5, 0.5, 1) для различия с танком игрока. Внутри TankBody добавил узел TankStats, который управляет характеристиками танка.

Шаг 4: Настройка характеристик танка

В узле TankStats добавил переменные: MaxHealth = 80.0 — максимальное здоровье AI-танка. BulletDamage = 8.0 — урон, который наносит снаряд. BulletSpeed = 450.0 — скорость снаряда. MovementSpeed = 0.9 — скорость передвижения танка.

Шаг 5: Настройка оружия танка

Узел TankWeapon, который управляет стрельбой AI-танка. Подключил переменную ProjectileScene, которая ссылается на сцену снаряда.

Шаг 6: Создание менеджера улучшений

Узел TankUpgradeManager, который управляет улучшениями танка. Подключил скрипт TankUpgradeManager.cs, чтобы реализовать логику улучшений.

Шаг 7: Создание менеджера игры

Узел GameManager, который управляет состоянием игры. Подключил скрипт GameManager.cs и добавил переменную ShapeScene, которая ссылается на сцену форм, которые игрок может уничтожать.

Шаг 8: Настройка границ арены

Узел ArenaBorder, который определяет границы игрового поля. Добавил узел Border типа Line2D для визуализации границ арены. Установил параметры, такие как width = 4.0 и default\_color = Color(0.3, 0.3, 0.3, 1) для визуального оформления границ.

Шаг 9: Создание интерфейса пользователя (HUD)

Узел HUD, который отображает информацию о здоровье, уровне и опыте игрока. Добавил элементы интерфейса: HealthLabel, LevelLabel, ExperienceBar. В методе \_Process обновляю текстовые метки и значения полос, используя переменные из TankStats, чтобы игрок мог видеть текущее состояние своего танка.

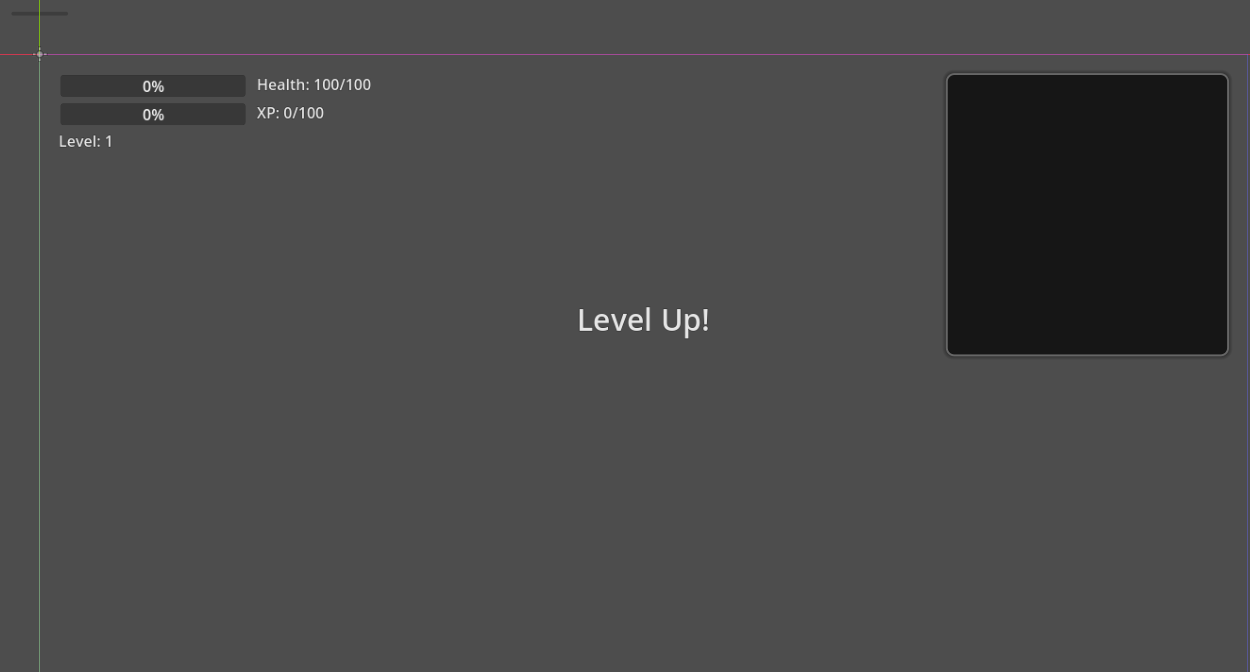


Рисунок 4 - HUD нтерфейса



Рисунок 5 - экран смерти

Шаг 10: Реализация системы улучшений

Узел UpgradeMenu, который позволяет игроку улучшать характеристики танка. Добавил кнопки: HealthUpgrade, ReloadUpgrade, DamageUpgrade. Каждая кнопка связана с методом, который вызывает UpgradeStat() в TankStats, чтобы увеличить соответствующую характеристику, если у игрока есть доступные очки улучшения.

Шаг 11: Управление снарядами

Узел для снарядов, чтобы управлять стрельбой из танка. В TankWeapon добавил переменную ProjectileScene, которая ссылается на сцену снаряда. В методе ShootAllCannons() создаю экземпляры снарядов и устанавливаю их свойства, такие как Speed, Damage и Penetration, используя значения из TankStats.

Шаг 12: Логика форм

Узел для форм, которые игрок может уничтожать. Каждая форма имеет переменные: Health и ExperienceValue. В методе TakeDamage() уменьшаю здоровье формы и, если оно достигает нуля, вызываю сигнал ShapeDestroyed.

Шаг 13: Интеграция компонентов

Обеспечил взаимодействие между компонентами, используя сигналы и соединения. Подключил сигнал pressed от кнопки RestartButton к методу OnRestartPressed() в HUD, чтобы перезапустить игру, когда игрок нажимает кнопку.

Шаг 14: Управление состоянием игры

Добавил переменные в GameManager, такие как IsGameOver и CurrentLevel. Реализовал метод CheckGameOver(), который проверяет, достиг ли игрок нуля здоровья, и вызывает соответствующие события для обновления интерфейса пользователя.

Шаг 15: Обработка уничтожения танка

В классе TankStats реализовал метод Die(), который обрабатывает логику уничтожения танка. При уничтожении танка вызывается сигнал TankDestroyed, который уведомляет другие компоненты о том, что танк был уничтожен, и обновляет состояние игры.

Шаг 1: Создание узла Tank

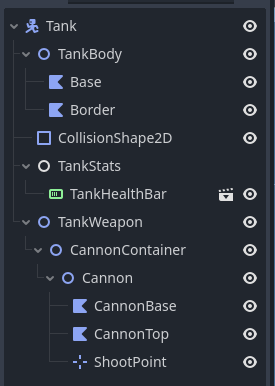
Узел Tank, который представляет собой основной танк игрока. Этот узел наследует от CharacterBody2D, что позволяет ему использовать физику и управление движением.

Добавил переменные:

collision\_layer = 2 — определяет, на каком слое находится танк, что позволяет ему взаимодействовать с другими объектами.

collision\_mask = 13 — определяет, с какими слоями танк может взаимодействовать, включая формы и AI-танки.

script = ExtResource("1\_2k4m3") — подключает скрипт TankController.cs, который управляет логикой танка.



Шаг 2: Настройка тела танка

Узел TankBody, который представляет собой визуальную часть танка. Этот узел будет содержать все визуальные элементы танка.

Добавил узел Base типа Polygon2D, который представляет основную форму танка.

color = Color(0.2, 0.4, 0.8, 1) — устанавливает цвет танка, чтобы он был легко различим.

polygon = PackedVector2Array(...) — определяет форму танка с помощью массива вершин, создавая его визуальный контур.

Добавил узел Border типа Polygon2D для создания границы танка.

z\_index = -1 — устанавливает порядок отрисовки, чтобы граница отображалась под основным телом танка.

color = Color(0.15, 0.3, 0.6, 1) — устанавливает цвет границы.

polygon = PackedVector2Array(...) — определяет форму границы танка.

Добавил узел CollisionShape2D для обработки столкновений.

shape = SubResource("CircleShape2D\_1") — устанавливает форму столкновения, используя круг для определения области, в которой танк может взаимодействовать с другими объектами.

Шаг 3: Настройка характеристик танка

Узел TankStats, который управляет характеристиками танка, такими как здоровье и урон.

script = ExtResource("2\_y6k4n") — подключает скрипт TankStats.cs, который содержит логику управления статистикой танка.

Шаг 4: Настройка полосы здоровья

Добавил узел TankHealthBar, который отображает текущее здоровье танка.

parent = TankStats — устанавливает родительский узел для полосы здоровья, чтобы она была связана с характеристиками танка.

instance = ExtResource("5\_projectile") — ссылается на сцену с визуальным представлением полосы здоровья.

Шаг 5: Настройка оружия танка

Узел TankWeapon, который управляет стрельбой танка.

type = Node2D — определяет, что это узел 2D, который будет содержать логику стрельбы.

rotation = -1.5708 — устанавливает начальный угол поворота оружия.

script = ExtResource("3\_l5m6n") — подключает скрипт TankWeapon.cs, который управляет стрельбой.

ProjectileScene = ExtResource("5\_projectile") — ссылается на сцену снаряда, который будет стрелять танк.

Шаг 6: Настройка контейнера для орудий

Добавил узел CannonContainer, который будет содержать все орудия танка.

parent = TankWeapon — устанавливает контейнер как дочерний элемент для узла оружия.

Шаг 7: Настройка орудий

Добавил узел Cannon, который представляет собой само орудие танка.

parent = TankWeapon/CannonContainer — устанавливает орудие как дочерний элемент контейнера орудий.

Добавил узел CannonBase типа Polygon2D для визуализации основания орудия.

z\_index = -1 — устанавливает порядок отрисовки, чтобы основание отображалось под верхней частью орудия.

color = Color(0.15, 0.3, 0.6, 1) — устанавливает цвет основания орудия.

polygon = PackedVector2Array(...) — определяет форму основания орудия.

Добавил узел CannonTop типа Polygon2D для визуализации верхней части орудия.

color = Color(0.2, 0.4, 0.8, 1) — устанавливает цвет верхней части орудия.

polygon = PackedVector2Array(...) — определяет форму верхней части орудия.

Добавил узел ShootPoint типа Marker2D, который указывает точку, откуда будут выстреливаться снаряды.

position = Vector2(0, -48) — устанавливает позицию точки стрельбы относительно орудия.

Шаг 8: Интеграция компонентов

Обеспечил взаимодействие между компонентами, используя сигналы и соединения. Это позволяет различным частям системы танка взаимодействовать друг с другом, например, когда танк получает урон или когда происходит стрельба.

Шаг 9: Реализация логики стрельбы

В классе TankWeapon реализовал методы, которые управляют стрельбой из орудий, включая создание снарядов и их свойства, такие как скорость и урон.

Шаг 10: Управление состоянием здоровья

В классе TankStats реализовал методы, которые управляют состоянием здоровья танка, включая восстановление здоровья и обработку получения урона.

Шаг 11: Обработка уничтожения танка

В классе TankStats реализовал метод, который обрабатывает уничтожение танка, включая логику начисления опыта игроку за уничтожение AI-танка.

Шаг 12: Управление улучшениями

В классе TankUpgradeManager реализовал логику управления улучшениями танка, включая возможность повышения уровня и изменения класса танка.

Шаг 13: Логика AI-танков

В классе AITankController реализовал логику поведения AI-танков, включая обнаружение игрока, преследование и отступление.

Шаг 14: Управление формами

В классе GameManager реализовал логику управления формами, которые игрок может уничтожать, включая спавн форм и их уничтожение.

Шаг 15: Обновление интерфейса пользователя (HUD)

В классе HUD реализовал логику отображения информации о здоровье, уровне и опыте игрока, чтобы игрок мог видеть текущее состояние своего танка.

9. Сигналы

Сцена включает в себя сигналы, которые обрабатывают взаимодействия пользователя. Например, кнопка перезапуска вызывает метод для перезапуска игры, а кнопки улучшений вызывают методы для применения соответствующих улучшений. Это обеспечивает интерактивность и отзывчивость интерфейса, позволяя игрокам легко взаимодействовать с элементами управления.

10. Ресурсы

Сцена ссылается на различные внешние ресурсы, включая скрипты, упакованные сцены и темы, которые определяют поведение и внешний вид узлов. Это позволяет легко управлять ресурсами и изменять их при необходимости. Использование внешних ресурсов также способствует модульности и повторному использованию кода, что упрощает разработку и поддержку проекта.

Заключение

Эта игра организована для обеспечения комплексного игрового опыта, включая управление игрой, управление игроком, пользовательский интерфейс для отображения здоровья и опыта, а также механики улучшения. Использование узлов, таких как CanvasLayer и Control, гарантирует, что пользовательский интерфейс остается отзывчивым и визуально отличимым от игрового мира. В результате игроки получают увлекательный и динамичный игровой процесс, который поддерживает интерес и вовлеченность.